Multiform Logical Time & Space for Mobile Cyber-Physical System with Automated Driving Assistance System

Qian Liu, Robert de Simone, Xiaohong Chen, Jing Liu

Multiform Logical Time & Space for Mobile Cyber-Physical System with Automated Driving Assistance System

Qian Liu*, Robert de Simone†, Xiaohong Chen*, and Jing Liu*
*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China
qianliu@stu.ecnu.edu.cn, xhchen@sei.ecnu.edu.cn, jliu@sei.ecnu.edu.cn
†INRIA Sophia Antipolis Méditerranée, Sophia Antipolis Cedex, France
Robert.de_Simone@inria.fr

***Abstract*—We study the use of Multiform Logical Time, as embodied in Esterel/SyncCharts and Clock Constraint Specification Language (CCSL), for the specification of assume-guarantee constraints providing safe driving rules related to time and space, in the context of Automated Driving Assistance Systems (ADAS). The main novelty lies in the use of logical clocks to represent the epochs of specific area encounters (when particular area trajectories just start overlapping for instance), thereby combining time and space constraints by CCSL to build safe driving rules specification. We propose the safe specification pattern at high-level that provide the required expressiveness for safe driving rules specification. In the pattern, multiform logical time provides the power of parameterization to express safe driving rules, before instantiation in further simulation contexts. We present an efficient way to irregularly update the constraints in the specification due to the context changes, where elements (other cars, road sections, traffic signs) may dynamically enter and exit the scene. In this way, we add constraints for the new elements and remove the constraints related to the disappearing elements rather than rebuild everything. The multi-lane highway scenario is used to illustrate how to irregularly and efficiently update the constraints in the specification while receiving a fresh scene.**

***Index Terms*—Multiform Logical Time, Space, Autonated Driving, Safety, Specification**

I. Introduction

Automated Driving Assistance System (ADAS) [1] is a typical application of the Mobile Cyber-Physical System (MCPS) [2]. In ADAS, there are three typical steps: *perception*, *plan*, and *act*. ADAS employs various sensors at any time when the car is running to *perceive* the surrounding environment, collect data, identify, detect, and track static and dynamic objects, thereby providing safe driving trajectory plan for autonomous vehicles. Then vehicles enforce the trajectory plan to effectively avoid traffic accidents. Unfortunately, many traffic accidents occur resulting from mistaken commands or operations provided by ADAS in recent years. For example, Uber's full vehicle automation caused a pedestrian death in 2017 [3]. Thus, it's very important to ensure the safety of driving rules for autonomous vehicles.

As safe driving rules are related to time and space, it is difficult for safety insurances. There are some researches on the specification considering both space and time properties. Most of them are spatial-temporal logics such as Spatio-temporal Logic for closure space (STLCS) [4], Spatio-temporal logical with S4u and Temporal Logical (STL) [5], and Signal Spatio-Temporal Logic (SSTL) [6]. These logics are all obtained by extending temporal logic or spatial logic.

Low-level safety assurances are also presented. For example, recently, Responsibility-Sensitive Safety (RSS), a white-box, interpretable, and mathematical model is proposed by Mobileye [7] to validate the constraints of safe driving rules at the low-level physical simulator, such as AirSim [8], Carla [9], Apollo [10], or Webots/SUMO [11]. However, due to the mobility of autonomous vehicles, elements (other cars, road pieces, traffic signs) may dynamically enter and exit the scene, the logic and low-level safety assurances can not deal with dynamically changing constraints in a unified way.

In this paper, we present a safety specification pattern, a high-level abstraction of the safe driving rules specifications, to build the specification for different scenarios, the framework as shown in Figure 1. The pattern is instantiated while receiving the fresh scene provided by the low-level physical simulator. Due to the changing scenes, the constraints in specifications are dynamically updated through time. With the specification, the observer could monitor the trajectory plan, and output whether it is safe or not. The specification consists of time and space constraints. In terms of space constraints, we present area trajectory (the area moves through continuous-time), and the spatial events generated by interactions between area trajectories. For time constraints, we adopt Clock Constraint Specification Language (CCSL) [12], [13], [14] that provides multiform logical time [15] as the parameters. Moreover, CCSL relies on the notion of logical clocks [16], which are commonly used to specify both partial orders and causal relationships on events, thus we can combine time and space constraints. In case of the pattern, three formalisms are

This work is supported by funding under National Key Research and Development Project 2017YFB1001800, NSFC 61972150, as well as Shanghai Science and technology program under grant No. 20ZR1416000. Corresponding authors are Jing Liu and Robert de Simone.

given and discussed, including Esterel/SyncCharts [17], Linear Temporal Logic [18] and CCSL.

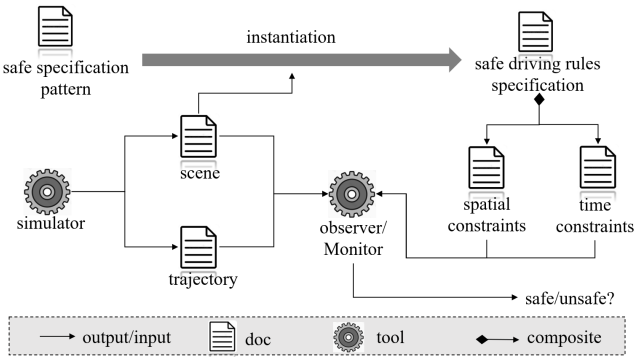


Fig. 1: Overview of our approach

To sum up, our contributions are as follows:

- We present a safe specification pattern to build specifications for safe driving rules, thereby providing a general approach to build specifications for different automated driving scenarios.
- In terms of space and time constraints in the specification, we introduce the notion of the spatial events, which are generated by the interaction between areas varying through time, so that combining them by CCSL. Additionally, we put forward two CCSL extensions to express safety property on the bounded future.
- An efficient approach is proposed to irregularly update the constraints during the specification generation by removing the constraints related to the disappearing elements or adding the new constraints for the new elements rather than rebuilding everything. Besides, we implement the spatial events and areas at the low-level physical simulator based on Responsibility Sensitive Safety.

The rest of this paper is organized as follows. Section II introduces the notations for Space and CCSL. Section III proposes safety specification pattern-based language. Section IV provides a specification generation process by instantiating the specification pattern. Section V presents an additional typical case to show how to instantiate the specification pattern for different scenarios. Section VI compares with the related work. Finally, Section VII concludes this paper and puts forward future work.

II. Notation for Space and Time in the Specification

In this section, we first put forward the notation for spatial events to build space constraints, then present the syntax and semantics of CCSL based on [14] and some extensions, thereby combining time and space constraints by CCSL.

A. Notations for Space

The safe driving rules specification involves the elements in the scene, and we need to consider them when constructing the specifications. In our setting, a *scene* consists of a certain number of $players$ (active elements, such as cars, pedestrian, bikers), $objects$ (passive elements, for instance, road lanes, crossings, traffic signs), and a specific "ego car", as shown in Definition 1.

Figure 2 presents two scenes a, b of the multi-lane highway (the solid red rectangle is the view of the ego car, and the dotted rectangles are the views of each direction), a is the previous scene of b. In this case, the $players$ only have vehicles, and the $objects$ only have road lanes. Thus, we express the scene a and b as

$a = \langle\{c_1, c_2, c_4, c_5, c_7\}, \{lane\_a, lane\_b, lane\_c\}, ego\rangle$,
$b = \langle\{c_1, c_2, c_6, c_7\}, \{lane\_a, lane\_b, lane\_c\}, ego\rangle$.

Definition 1: (**Scene**). The scene is a triple $\langle players, objects, ego\rangle$, where $players$ is identifiers set of active elements, and $objects$ is identifiers set of passive elements, ego is the car itself.

For each of these generic element types, we define several areas of relevance for safety. In terms of the ego car, it includes 1) $front\ collision\ area$ 2) $front\ danger\ area$ 3) $front\ view\ area$, and similarly for right, left, and back areas, as shown in Definition 2. If the danger area intersects other cars/pedestrians, then the ego car enters into a dangerous state. Once other cars/pedestrians overlap the collision area, the ego car happens a collision. The view of ego car is the area the ego car can perceive. To express the area of the $players$, we present a $Near$ operator, which has different definitions for different elements.

Combining with Definition 1, we can see that the $players$ are the areas of active elements that intersect or belong to the perception area of the ego car, the $objects$ are the areas of passive elements that intersect or belong to the perception area of the ego car.

The areas in the multi-lane highway are as shown in Figure 2 and 3. The former presents the view areas of ego (left view, right view, back view, and front view). The latter shows the danger area and collision area of ego, and the $Near$ area of $players$ and $objects$. Thus, the $players$ and $objects$ of ego car in each direction are defined as

$players_{direction}(ego) = \{id \in players \mid id \sqcap view_{direction}(ego)\}$
$objects_{direction}(ego) = \{id \in objects \mid id \sqcap view_{direction}(ego)\}$,

where $direction \in \{left, right, back, front\}$. Specially, given an area $A \in view(ego)$, the $players$ of ego car in A denoted as $players(A, ego)$, the $objects$ of ego car in A denoted as $objects(A, ego)$. In the scene a as shown in Figure 2(a), the $players$ of ego car in each direction are in the following.

- $players_{left}(ego) = \{c_1, c_2\}$,
- $players_{right}(ego) = \{c_7\}$,
- $players_{front}(ego) = \{c_2, c_5, c_7\}$,
- $players_{back}(ego) = \{c_1, c_4\}$.

In the scene a as shown in Figure 2(b), the $players$ of ego car in each direction are in the following.

- $players_{left}(ego) = \{c_1, c_2\}$,
- $players_{right}(ego) = \{c_6, c_7\}$,
- $players_{front}(ego) = \{c_2, c_7\}$,
- $players_{back}(ego) = \emptyset$.

As for areas of the ego car, Figure 3 shows the collision area

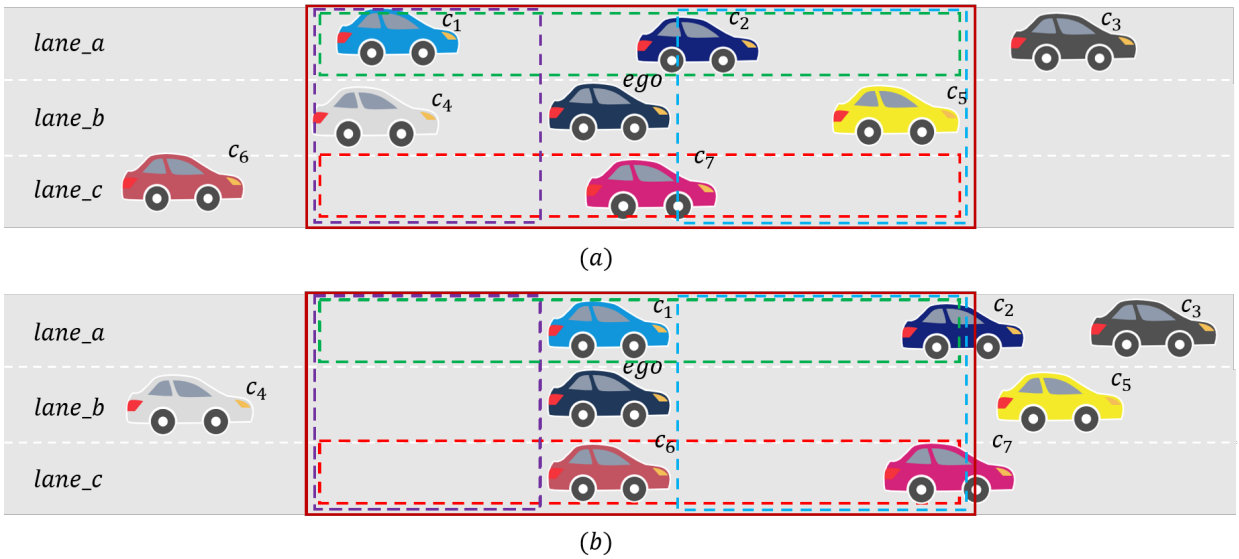


Fig. 2: The multi-lane highway scenario, where a and b are scenes, a is the previous one of b. a is the previous scene of b. There are three road lanes $lane\_a, lane\_b, lane\_c$ and eight vehicles $ego, c_1, c_2, c_3, c_4, c, c_6, c_7$, and the ego car drives along the middle lane $lane\_b$. The solid red rectangle is the view of the ego car, and dotted rectangles are the views of each direction (the blue one is front view, the red one is right view, the green one is left view, and the purple one is back view).

of the ego car is the space occupied by the ego car. The danger area is defined by the longitudinal and lateral safe distance [7]. The longitudinal safe distance is for the front and back danger area, and the lateral one is for the right and left. And the $Near$ area of the $players$ is the area occupied by the $players$.

Definition 2: (**Areas in the scene**). The syntax of areas in the scene is defined as

$$area ::= id \in \mathbb{I} \mid ego \mid type_{direction}(ego) \mid Near(id),$$

where $\mathbb{I}$ is a countably infinite set of globally unique identifiers, $type \in \{collision, danger, view\}$, and $direction \in \{left, right, back, front\}$. $Near(id)$ depends on the element id, which has different definitions for different elements.

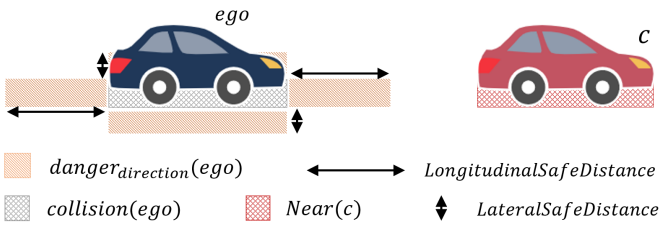


Fig. 3: Areas in the scene, where $direction \in \{left, right, back, front\}$, the longitudinal and lateral safe distances are from RSS [7].

Then we present area trajectories in Definition 3 as these areas move through continuous-time in simulation. Specially, given the area A and time point $t \in \mathbb{R}^+$, $A(t)$ is the area A at time t.

Definition 3: (**Area Trajectory**). Given an area A, the area trajectory of A is a total function $A : \mathbb{R}^+ \to 2^{\mathbb{U}}$, where $\mathbb{U}$ is the universe space, $\mathbb{R}^+$ represents the continuous (physical) time.

Definition 4: (**Spatial Relations**). The syntax of spatial relations is defined as $\gamma ::= A \triangleq B \mid A \sqsubseteq B \mid A \ominus B \mid A \sqcap B$.

To express the relation between areas, Definition 4 presents the spatial binary relations, including equal ($\triangleq$), belongs to ($\sqsubseteq$), disjoint ($\ominus$), intersection ($\sqcap$). The semantics of them are in the following, as the same as Figure 4, where $A, B \subseteq \mathbb{U}$.

- $A \triangleq B$ means that any point in B is in A, vice versa, that is formulated as
$$\forall p \in \mathbb{U}, p \in A \Longleftrightarrow p \in B;$$
- $A \sqsubseteq B$ means that any point in B is in A, but unnecessarily vice versa, that is formulated as
$$\forall p \in \mathbb{U}, p \in B \Longrightarrow p \in A;$$
- $A \ominus B$ means that any point in B is not in A, vice versa, that is formulated as
$$\forall p \in \mathbb{U}, \neg(p \in B \wedge p \in A);$$
- $A \sqcap B$ means that there exists some points in B and A, that is formulated as
$$\exists p \in \mathbb{U}, p \in A \wedge p \in B.$$

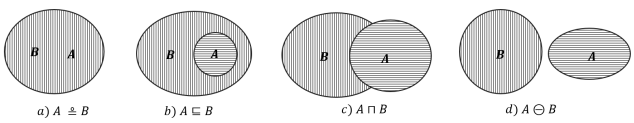


Fig. 4: Semantic of spatial relations

Also, Definition 5 presents spatial events, generated by the interaction between area trajectories, to express transition between the relations. Spatial events includes *join*, *disjoin*, *include*, *exclude*. The spatial events are used to express the transitions between the relations, as shown in Figure 5.

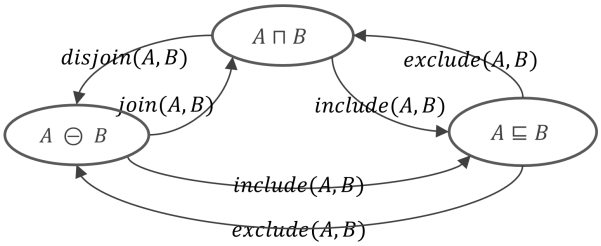


Fig. 5: Transitions between spatial relations

Definition 5: (**Spatial Events**). The syntax of spatial events is defined as

$$e ::= join(A,B)|disjoin(A,B)|include(A,B)|exclude(A,B),$$

where A and B are the areas and vary with the time. We present the semantics of these events in the following, where $\epsilon \in \mathbb{R}^+$ is small enough.

- $join(A,B)$ is used to express the transition from relation $A \ominus B$ to $A \sqcap B$, that is formulated as

$$join(A,B) = \{t \in \mathbb{R}^+ | A(t) \ominus B(t) \wedge A(t+\epsilon) \sqcap B(t+\epsilon)\};$$

- $disjoin(A,B)$ is used to express the transition from relation $A \sqcap B$ to $A \ominus B$, that is formulated as

$$disjoin(A,B) = \{t \in \mathbb{R}^+ | A(t) \sqcap B(t) \wedge A(t+\epsilon) \ominus B(t+\epsilon)\};$$

- $include(A,B)$ is used to express the transition from relation $A \sqcap B$ or $A \ominus B$ to $A \sqsubseteq B$, formulated as

$$include(A,B) = \{t \in \mathbb{R}^+ | (A(t) \sqcap B(t) \vee A(t) \ominus B(t)) \wedge A(t+\epsilon) \sqsubseteq B(t+\epsilon)\};$$

- $exclude(A,B)$ is used to express the transition from relation $A \sqsubseteq B$ to $A \sqcap B$ or $A \ominus B$, formulated as

$$exclude(A,B) = \{t \in \mathbb{R}^+ | A(t) \sqsubseteq B(t) \wedge (A(t+\epsilon) \sqcap B(t+\epsilon) \vee A(t+\epsilon) \ominus B(t+\epsilon))\}.$$

TABLE I: Syntax and semantics of spatial relations and events, where area A and B vary with the time, $\epsilon \in \mathbb{R}^+$ is small enough.

Spatial Relations	$1. A \triangleq B \iff \forall p \in \mathbb{U}, p \in A \iff p \in B$
	$2. A \sqsubseteq B \iff \forall p \in \mathbb{U}, p \in B \implies p \in A$
	$3. A \ominus B \iff \forall p \in \mathbb{U}, \neg(p \in B \wedge p \in A)$
	$4. A \sqcap B \iff \exists p \in \mathbb{U}, p \in A \wedge p \in B$
Spatial Events	$5. join(A,B) = \{t \in \mathbb{R}^+ \| A(t) \ominus B(t) \wedge A(t+\epsilon) \sqcap B(t+\epsilon)\}$
	$6. disjoin(A,B) = \{t \in \mathbb{R}^+ \| A(t) \sqcap B(t) \wedge A(t+\epsilon) \ominus B(t+\epsilon)\}$
	$7. include(A,B) = \{t \in \mathbb{R}^+ \| (A(t) \sqcap B(t) \vee A(t) \ominus B(t)) \wedge A(t+\epsilon) \sqsubseteq B(t+\epsilon)\}$
	$8. exclude(A,B) = \{t \in \mathbb{R}^+ \| A(t) \sqsubseteq B(t) \wedge (A(t+\epsilon) \sqcap B(t+\epsilon) \vee A(t+\epsilon) \ominus B(t+\epsilon))\}$

We summarize the syntax and semantics of spatial relations and events in Table I. In the multi-lane highway, the spatial events of the ego car are as shown in the following.

(1) $join(danger_{direction}(ego), Near(c))$
(2) $disjoin(danger_{direction}(ego), Near(c))$
(3) $join(collision(ego), Near(c))$,

where $direction \in \{left, right, back, front\}$, and $c \in players_{direction}(ego)$; $join(danger_{direction}(ego), Near(c))$ means the distance between ego car and other cars from "greater than the safe distance" to "less than the safe distance" in each direction; $disjoin(danger_{direction}(ego), Near(c))$ says the distance between ego car and other cars from "less than the safe distance" to "greater than the safe distance" in each direction; $join(collision(ego), Near(c))$ states the ego car happens a collision.

B. Notation for CCSL

CCSL relies on the notion of logical clocks, which are commonly used to specify both partial orders and causal relationships on events. Based on spatial events defined in II-A, we can combine time and space constraints by CCSL. Logical clocks are infinite sequences of ticks as shown in Definition 6.

Definition 6: (**Logical clock**). A logical clock c is defined as an infinite sequence of ticks: $(c_n)_{n=1}^{\infty}$.

Clocks describe events noticeably in a system. Their ticks denote their occurrences. During the execution of a system, clocks tick according to occurrences of related events. In Definition 7, the schedule is to capture what occurs during a specific execution period.

Definition 7: (**Schedule**). Given a set C of clocks, a schedule of C is a total function $\delta : \mathbb{N}^+ \to 2^C$ such that at each step n in $\mathbb{N}^+$, $\delta(n) = \{c | c \in C \wedge c_n = tick\}$ and $\delta(n) \neq \emptyset$.

Schedule is an infinite sequence consisting of a set of ticked clocks at any steps, where $\delta(n)$ represents the set of ticked clocks at step n. Clock $c \in \delta(n)$ means that clock c ticks at the n^{th} step. And $\delta(n) \neq \emptyset$ means that there is at least a clock ticked at any steps of schedule. To model general clock constraints in CCSL, we adopt the notion of history that decides what may have at a given step in Definition 8.

Definition 8: (**History**). A history of a schedule δ over a set C of clocks is a function $\chi_\delta : C \times \mathbb{N}^+ \to 2^C$ such that for each clock $c \in C$ and $n \in \mathbb{N}^+$:

$$\chi_\delta(c,n) = \begin{cases} 0 & if \quad n = 1 \\ \chi_\delta(c, n-1) & if \quad n > 1 \wedge c \in \delta(n-1) \\ \chi_\delta(c, n-1) + 1 & if \quad n > 1 \wedge c \notin \delta(n-1) \end{cases}$$

History records the number of clock ticks. For any $n \in \mathbb{N}^+$, the number of ticks of $\chi_\delta(c,n)$ clock c before step n. In particular, at the first step $\chi_\delta(c,1) = 0$.

A CCSL specification consists of a set of constraints, where each constraint expresses the relations between the ticks of two clocks. Table II lists the eleven primitive CCSL operations that can be used to compose all the possible constraints, where the top five operators (coincidence($\triangleq$), precedence($\prec$), causality($\preceq$), subclock($\subseteq$), exclusion($\#$)) can model relation constraints, and the remaining six operators (union($+$), intersection($*$), infimum($\wedge$), supremum($\vee$), delay($\$$), periodicity($\propto$)) can model expression constraints.

TABLE II: Syntax and semantic of CCSL, where c_1, c_2, and c_3 are logical clock (events), δ is the schedule of them.

Syntax	Semantics
$\delta \vDash c_1 \triangleq c_2$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) = \chi_\delta(c_2, n)$
$\delta \vDash c_1 \prec c_2$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) = \chi_\delta(c_2, n) \Rightarrow c_2 \notin \delta(n)$
$\delta \vDash c_1 \preceq c_2$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) \geq \chi_\delta(c_2, n)$
$\delta \vDash c_1 \subseteq c_2$	$\forall n \in \mathbb{N}^+, c_1 \in \delta(n) \Rightarrow c_2 \in \delta(n)$
$\delta \vDash c_1 \# c_2$	$\forall n \in \mathbb{N}^+, c_1 \notin \delta(n) \vee c_2 \notin \delta(n)$
$\delta \vDash c_1 \triangleq c_2 + c_3$	$\forall n \in \mathbb{N}^+, c_1 \in \delta(n) \Leftrightarrow c_2 \in \delta(n) \vee c_3 \in \delta(n)$
$\delta \vDash c_1 \triangleq c_2 * c_3$	$\forall n \in \mathbb{N}^+, c_1 \in \delta(n) \Leftrightarrow c_2 \in \delta(n) \wedge c_3 \in \delta(n)$
$\delta \vDash c_1 \triangleq c_2 \wedge c_3$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) = max[\chi_\delta(c_2, n), \chi_\delta(c_3, n)]$
$\delta \vDash c_1 \triangleq c_2 \vee c_3$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) = min[\chi_\delta(c_2, n), \chi_\delta(c_3, n)]$
$\delta \vDash c_1 \triangleq c_2 \$ d$	$\forall n \in \mathbb{N}^+, \chi_\delta(c_1, n) = max[\chi_\delta(c_2, n) - d, 0]$
$\delta \vDash c_1 \triangleq c_2 \propto p$	$\forall n \in \mathbb{N}^+, c_1 \in \delta(n) \Leftrightarrow c_2 \in \delta(n) \wedge \exists m \in \mathbb{N}^+, \chi_\delta(c_2, n) = m \times p - 1$

CCSL Extensions. To build constraints on the bounded future, we need CCSL to express more complex time-related requirements such as "within some seconds", "after some seconds", and "after some meters". We thus present two CCSL extensions as shown in the following.

- $c_1 \triangleq c_2$ *within* d, where d is multiform logical time, means if c_2 ticks, then the next tick of c_1 should tick within d time units.

- $c_1 \triangleq c_2$ *after* d, where d is multiform logical time, means if c_2 ticks, the next tick of c_1 should tick after d time units.

III. Safe Specification Pattern

By introducing spatial events and CCSL extensions, we can use CCSL to build a safe driving specification. This section introduces the safe specification pattern to build specifications dynamically for different scenarios, which is the higher-level abstraction of safe driving rules. Additionally, we use three formalism language to build expressions for the pattern, then discuss their differences.

A. Pattern expression and formalism

In the specification, the spatial and time constraints are all expressed by CCSL expressions, which are based on events. Thus, we present a safe specification pattern, as shown in Table III. In the pattern, there are three template events (logical clock), $StartCheck$, $Recover$, and $Failure$. It states that once a $StartCheck$ temporal condition is detected, initializing the realization that there might be a latent problem to be monitored, then one awaits for any of two subsequent events: a positively resolving $Recover$ event, stating that things went back to normal, or a catastrophic $Failure$ event, notifying a potential collision of a timed-out delay for remaining too long in the problematic model.

We considered three formalism languages: CCSL, Esterel/SyncCharts, and LTL. While they may roughly all be amenable to translation into state-based *observers/monitors*, they provide different specification styles, each with its quality for the natural expression of constraints in certain cases. We shall comment at length on the comparison and commonalities

TABLE III: Safety specification pattern and expression of Esterel, CCSL, and LTL

Safe Specification Pattern	
Once *StartCheck* temporal condition is detected, a monitor is initialized to detect possible latent problems. The monitor stops either with a positive issue, when receiving the *Recover* event, or a negative one, when receiving a *Failure* event (collision or time-out, for instance).	
Formalism	**Expression**
Esterel	**await** $StartCheck$ **then (await** $Recover$**) abort** $Failure$
CCSL	$StartCheck$ **alternatesWith** $Recover$ $Failure = 0$ *(the never-ticking clock)*
LTL	**always (**$StartCheck$ **implies (not** $Failure$**)) until** $Recover$

of these 3 languages below. We shall not be overly complete, as our focus in this paper is more on the concrete usefulness of possible fragments of these specification languages in our context.

While we overly loaded our three template events with subjective interpretation, it may be adapted to each particular specification case. Through careful analysis, we obtain the three formalism expression for the specification pattern in Table III:

- In CCSL constraints, it states either *StartCheck* or *Recover* ticks and *StartCheck* ticks before *Recover*, *Failure* never ticks.
- In Esterel program, it means once *StartCheck* occurs, then from this moment wait and check which of the two events *Failure* or *Recover* occurs.
- In LTL, it says once *StartCheck* occurs, then *Failure* never occurs until *Recover* occurs.

Below, we build specification for multi-lane highway, as shown in the following.

$\forall direction \in \{left, right, back, front\}$,

$\forall c \in players_{direction}(ego)$,

- $StartCheck = join(danger_{direction}(ego), Near(c))$
- $Recover = disjoin(danger_{direction}(ego), Near(c))$

$$\triangleq join(danger_{direction}(ego), Near(c)) \; within \; max\_resp\_time,$$

- $Failure = join(collison(ego), Near(c))$

which states that in each direction once the danger area of ego car intersects with another car (or with other cars), we expect a recovery within a given max response time to prevent a collision. In this specification, "the danger area of ego car intersects with another car" means the distance between the ego car and other cars is less than the safe distance. So the "recovery" is the distance between the ego car and other cars greater than the safe distance within a max response time, otherwise, the ego car happens a collision.

B. Comparison

Here we discuss the differences between the three formalism expressions in Table III:

- Several successive *StartCheck*s should not occur before conclusive events (either *Recover* or *Failure*) occur for the first *StartCheck*. In this case, the Esterel/SyncCharts form reset

the monitoring to the latest start, while the LTL form applies possibly the conclusions of later starts to the earlier one. In fact here we simply want to disregard as irrelevant the cases where several *StartCheck* may occur before the previous one has been resolved. This is usually implicit in the designer's mind.

- For CCSL form, in general Events *StartCheck*, *Recover* and *Failure* may not be primitive, but instead themselves computed as temporal combinations of primitive input events. This can easily be realized in Esterel/SyncCharts or CCSL by adding constraints in parallel, and Esterel/SyncCharts allows in addition a local-state based specification style, universally popular owing to StateCharts and UML state machines. In contrast, LTL can encode states, but in a lengthy expansive way, and cannot record parallelism. In this respect, the CCSL specification style (which deals only with clocks/events as sequences, and not individual occurrences) is also less intuitive, as certainly shown on our example.

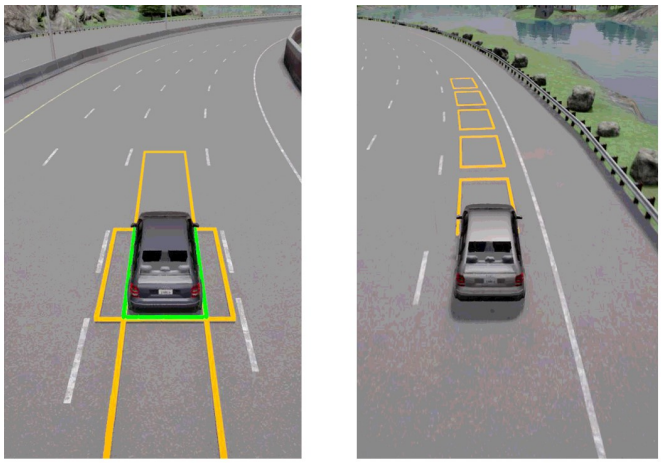
- One supposed interest of LTL is that it supports natively qualification of results (as true or false, meaning success or failure of the property monitored. This is also a weakness, as it does not allow flexibility when qualifying conclusions at the level of operational monitors. In the case of Esterel/SyncCharts extra mechanisms need to be specified for the monitors (interpretation as positive and negative answers respectively of the *ok* and *ko* events respectively. The case when a property is satisfied only "at infinity" (not eventually *p*) is also a subtle issue (as in monitoring a system it won't be checkable by a finite run anyhow). It shall come back to this later. In the case of CCSL, one can specify (by default) that any non-syntactically finite clock should indeed tick infinitely often (infinite sequence of tick occurrences).

IV. Specification Generation

Safe driving rules specification generation relies on the scene information (areas and spatial events defined in Section II-A) that provided by the low-level simulation. Due to mobility, the scene varies with time. Thus, we need irregularly update the constraints in the specification. In this section, we first show how to implement the areas and spatial events at a low-level physical simulator, thereby providing the inputs for specification generation. Then, we present an efficient way to dynamically update the constraints, finally simulate the processes of specification generation in the Carla simulator.

A. Areas and Spatial Events in low-level Simulator

Safe driving rules specification generation needs the low-level physical simulator to provide areas and area trajectories of ego car and the surrounding elements as the input, then the observer could monitor the trajectory plan and output whether it is safe or not. There are several mainstream simulators for self-driving, such as AirSim, Carla, Apollo, and Webots/-SUMO. In our work, we adopt Carla as the low-level physical simulator, which integrates the C++ Library for Responsibility Sensitive Safety (RSS-LIB) [19] in the client library, the areas of ego car based on RSS-LIB implement in Carla simulator as shown in Figure 6. Responsibility-Sensitive Safety (RSS) [7] is a white-box, interpretable, mathematical model for safety assurance of Self-driving. In this model, it introduces lateral and longitudinal safe distance, where lateral safe distance is for the left and right, the longitudinal one is for back and front. Here we don't show the definitions, which could be found in [7].



(a) areas of ego car **(b) area trajectory of ego car**

```
Spatial_Events.JOIN,Spatial_Events.INCLUDE,Spatial_Events.EXCLUDE,Sp
atial_Events.INCLUDE,Spatial_Events.EXCLUDE,Spatial_Events.INCLUDE,S
patial_Events.EXCLUDE,Spatial_Events.INCLUDE,Spatial_Events.EXCLUDE,
Spatial_Events.INCLUDE,Spatial_Events.EXCLUDE,Spatial_Events.DISJOIN
,Spatial_Events.JOIN,Spatial_Events.INCLUDE,Spatial_Events.EXCLUDE,S
patial_Events.INCLUDE,Spatial_Events.EXCLUDE,Spatial_Events.DISJOIN,
```

(c) spatial events generated by interaction on areas trajectories

Fig. 6: The visualization of areas and area trajectory of ego car and generation of spatial events in Carla simulator

In this paper, we use the longitudinal and lateral safe distance proposed in RSS-LIB to implement the danger areas of ego car. The collision area of the ego car is the space occupied by the ego car, which is defined as the rectangle, as shown in Figure 6(a). In the multi-lane highway, the $Near$ area of the players also is implemented by the rectangle, which is the space occupied by the player. These areas are implemented with Application Interface (API) provided by RSS-LIB.

With these areas implemented at low-level based on RSS-LIB, we can obtain the area trajectories of ego car and $players$, one area trajectory of ego car, as shown in Figure 6(b). Then we generate the spatial events by the interaction on area trajectories, they are sorted in order of occurrence, as shown in Figure 6(c). Based on these events, we can verify the safety of driving rules.

B. Constraints Update

When building the specification with the pattern, we need to irregularly update constraints in the specification. Traditionally, once receiving fresh scene information, ADAS rebuilds everything to generate the constraints, but it takes too many resources and inefficient. Below, we present an efficient way to irregularly update the constraints in safe driving rules specifications.

TABLE IV: The $players$ of ego car in each direction, where a is the previous scene of b.

expression	scene a	scene b
$player_{left}(ego)$	$\{c_1, c_2\}$	$\{c_1, c_2\}$
$player_{right}(ego)$	$\{c_7\}$	$\{c_6, c_7\}$
$player_{front}(ego)$	$\{c_2, c_5, c_7\}$	$\{c_2, c_7\}$
$player_{back}(ego)$	$\{c_1, c_4\}$	$\emptyset$

In our approach, we add the constraints related to the elements entering the view and remove the constraints related to the element exiting the view. In Algorithm 1, it shows how to update constraints in the specification while receiving a fresh scene. By building a table of the $players$ and $objects$ to preserve the previous and the current scene information, and comparing them from each direction, we could know which constraints need to be removed and for which element need to add constraints.

Algorithm 1: Update constraints in specification while receiving a fresh scene

Input:
The previous scene,
$a = \langle pre\_players, pre\_objects, ego \rangle$;
The fresh scene, $b = \langle players, objects, ego \rangle$;
for each $d \in \{left, right, back, front\}$ **do**
  for all $c \in players_d(ego)$ such that $c \notin pre\_players_d(ego)$ **do**
    Add the constraints of c for the direction d;
  end for
  for all $c \in pre\_players_d(ego)$ such that $c \notin players_d(ego)$ **do**
    Remove the constraints of c for the direction d;
  end for
  for all $c \in objects_d(ego)$ such that $c \notin pre\_objects_d(ego)$ **do**
    Add the constraints of c for the direction d;
  end for
  for all $c \in pre\_objects_d(ego)$ such that $c \notin objects_d(ego)$ **do**
    Remove the constraints of c for the direction d;
  end for
end for

For example, Figure 2 shows two scenes a and b of multi-lane highway, where a is the previous one of b. With Algorithm 1, we can update the constraints while receiving the scene b. Thus, a is the previous scene, b is the current scene. In these two scenes, we can see that the objects don't change, so in Table IV, it only shows the $players$ of ego car in each direction of scene a and b. From the table, we can see that the left $players$ of ego car don't change, so we needn't do anything for the constraints of the left direction. But in the right direction, a new vehicle c_6 enters the right view of ego car, we thus need to add the constraint of c_6 for the right direction. Also, in terms of the front direction, the vehicle c_5 exits the front view of ego car, we thus need to remove the constraints related to c_5 for the front direction. The all rear $players$ exit the back view of ego car, we thus remove all constraint for back direction. Therefore, while receiving the fresh scene b, ADAS need to enforce these operations to update the constraints as shown in the following.

(1) Add a constraint of c_6 for the right direction;
(2) Remove the constraint of c_5 for the front direction;
(3) Remove the constraint of c_1 and c_4 for the back direction.

C. Simulation

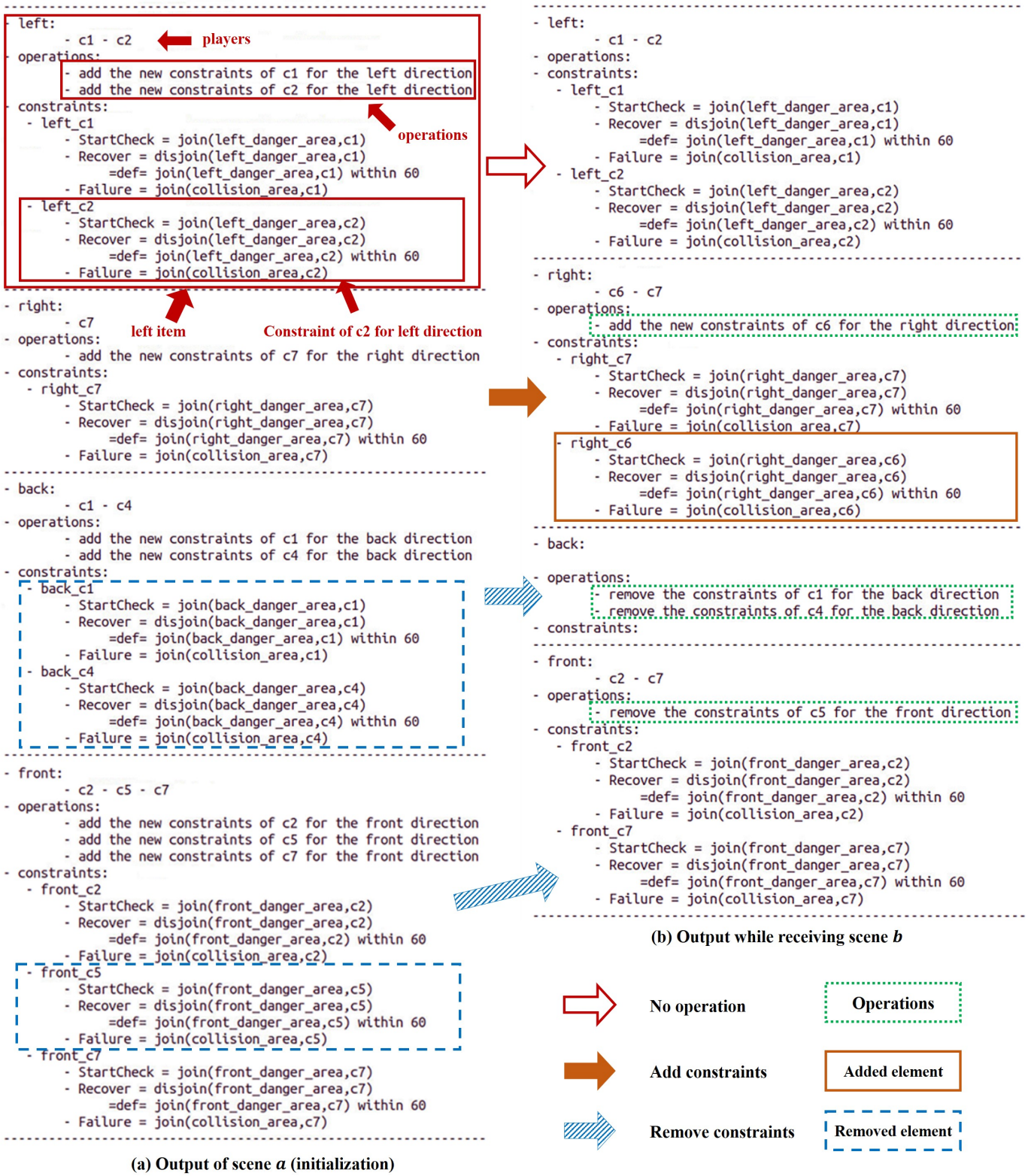
We implement Algorithm 1 using Python programming language, and simulate the ego car drives on the highway in Carla simulator (version 0.9.9, building on Ubuntu 18.04 LTS) which runs on Unreal Engine (version 4.24). In the simulation, we adopt the *Town04* (an infinite loop with a highway and a small town) as the map.

As for the multi-lane highway, we already build the specification, which takes the "direction" and "time" as parameters. In the simulation, the "directions" are right, left, back, and front, "time" is 60 seconds. Then, we simulate the process of updating constraints when receiving new scenes on a multi-lane highway, and the information related to updating constraints are saved in a $.log$ file. The structure of this file is shown in Figure 7. Figure 7(a) and Figure 7(b) are the basic elements. Each basic element corresponds to a scene, and there are four same structure items in the basic element, respectively to left, right, back, and front. Each item contains the corresponding $players$, $operations$, $constraints$, where $players$ are the vehicles nearby the ego car, $operations$ are performed when the constraint was last updated, $constraints$ is a set of constraint referring to $players$. In terms of the two scenes a and b as shown in Figure 2, we add constraints for each direction with the scene a. Figure 7 is the outputs of the simulation for a and b. Figure 7(a) presents the $players$, $constraints$, and $operations$ in each direction when the simulator is initializing with scene a. While receiving the scene b, the simulator computes the difference between a and b, and then proposes the operations to enforce, thereby updating the constraints, as shown in Figure 7(b).

V. Case Study: T-Junction

In this section, we build the safety specification for T-Junction) to show the effectiveness of our framework for different scenarios. In T-Junction, there are two roads. Each of them has a waiting area, denoted by $HighPriorityWaitArea$, $LowPriorityWaitArea$. They merge into one road, as shown in Figure 8. Assuming that the overlap area is the "only one car area" $OnlyOneCarArea$, which means anytime there is at most one car enters into this area. In this case, it should obey the traffic rules to avoid collisions. The safe driving rules are in the following.

- If ego car approaches the junction on the low-priority road, then ego car waits until there doesn't exist another



(a) Output of scene *a* (initialization)

(b) Output while receiving scene *b*

Fig. 7: The outputs of the simulation (from scene a to b),

car on the overlap area and high-priority road, then ego car moves to the junction.
- If ego car approaches from the high-priority road, it only checks (in normal conditions) whether there is another car in the overlap area, and need not check the low-priority road. Once there doesn't exist another car on the overlap area, the ego car moves to the junction, otherwise, it waits until there no exists other cars on the overlap area.

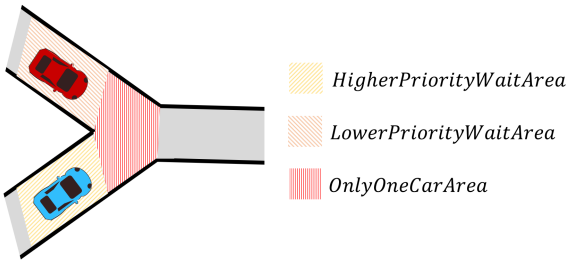


Fig. 8: Junction

In these two rules, the former is for the ego car drives on the low-priority road, another is for high-priority one. They all start to check $StartCheck$ when they arrive at their waiting area $join(ego, LowPriorityWaitArea)$, $join(ego, HighPriorityWaitArea)$. On the low-priority road, ego car needs to check the high-priority area and the overlap area, and ensure that the two areas are all not occupied by other cars. If there doesn't exist cars already in the high-priority area $include(another, HighPriorityWaitArea)$ and the overlap area pass, $exclude(anotherCar, OnlyOneCarArea)$, then the ego car enters into the junction $include(ego, OnlyOneCarArea)$. On high-priority road, ego car just check whether the overlap area is occupied by another car $anotherCar$, stating that after the car in the overlap area pass $exclude(anotherCar, OnlyOneCarArea)$, then ego car can pass the overlap area $include(ego, OnlyOneCarArea)$. Below, we present the safety specification to model the driving rules in this scenario, that are in the following.
(1) ego car on the low-priority road, if another car is in $HighPriorityWaitArea$ or $OnlyOneCarArea$, which indicates

$$players(HighPriorityWaitArea, ego) \neq \emptyset \, or$$
$$players(OnlyOneCarArea, ego) \neq \emptyset,$$

so the specification can be expressed as:

$$\forall\, anotherCar \in players(OnlyOneCarArea, ego) \cup players(HighPriorityWaitArea, ego),$$

- $StartCheck =$
$$(join(ego, LowPriorityWaitArea) + include(ego, LowPriorityWaitArea)) * (include(anotherCar, OnlyOneCarArea) + include(anotherCar, HighPriorityWaitArea))$$
- $Recover = exclude(anotherCar, OnlyOneCarArea);$
- $Failure =$
$$include(ego, OnlyOneCarArea) * include(another, OnlyOneCarArea).$$

(2) ego car on the high-priority road, if another car is in $OnlyOneCarArea$, which means

$$players(OnlyOneCarArea, ego) \neq \emptyset,$$

so the specification can be expressed as:

$$\forall\, anotherCar \in players(OnlyOneCarArea, ego),$$

- $StartCheck =$
$$(join(ego, HighPriorityWaitArea) + include(ego, HighPriorityWaitArea)) * include(anotherCar, OnlyOnCarArea);$$
- $Recover = exclude(anotherCar, OnlyOnCarArea);$
- $Failure =$
$$include(ego, OnlyOneCarArea) * include(anotherCar, OnlyOnCarArea).$$

Constraints update. In this case, while receiving a fresh scene, the ego car will use the same approach as the multi-lane highway to update the constraints. The difference between them is that T-junction updates the constraints with a specific part of the view, but multi-lane highway uses the whole view. As for the low-priority road, the partial views are the overlap area $OnlyOnCarArea$ and the high-priority waiting area $HighPriorityWaitArea$, and as for the high-priority road, it is the overlap area $OnlyOnCarArea$.

VI. Related Work

Safety is a key issue in self-driving automotive and Automated Driver Assistance Systems (ADAS). Responsibility-Sensitive Safety (RSS), proposed by Mobileye, introduces white-box mathematical modeling for safety assurance [7]. While spatial/topological and physical (low-level) aspects are handled, a precise formal language expressing constraints and properties of the assumed trajectory planning is missing, although its need is mentioned. A partial answer is provided in [20], aiming to monitor RSS rules by encoding signal temporal logic (STL); again this work lacks a precise identification of zero-crossing events, as well as constraint modularity.

Signal temporal logic (STL) [21] is designed to handle continuous/dense time signals and describe behaviors of continuous and hybrid systems. Timestamp Temporal Logic (TTL) [22] advocates a cleaner treatment of zero-crossing events abstracting from STL. The Zelus language [23] also introduces appropriate notations to "jump" from physical to logical time, this time for programming purposes. Our contribution here is to link explicitly timed_events/logical_clocks to dedicated spatial encounters, that can readily be used indirectly readable constraint in that domain; also, our formalism can be used for assumptions/provisions as well as guarantee/requirements.

The Clock Constraint Specification Language (CCSL) [12], [13], [24], [14] is an annex of MARTE [25] specification to elaborate and reason on the logical time model [26]. It

has a strong ability to express time constraints. Based on CCSL, many contributions have been introduced to analyze schedulability and generate efficiency scheduling [27], [28], [14]. Some studies adopt CCSL to express the time properties for the Spatio-temporal system [29]. Recently, [30] extends CCSL (PrCCSL) to formal verification of dynamic and stochastic behaviors for Automotive Systems, which lacks how to express the space constraints. In this paper, we define spatial events generated by interaction on area trajectories, thereby combining space and time constraints with CCSL and improving the expressiveness ability of CCSL.

Finally, the STIMULUS tool by ArgoSim (now Dassault System) [31] provides specific patterns to easily synthesize properties and extract monitoring observers from them. In a way, this is perhaps the closest relation to our work. Still, the precise means to connect our constraints to basic events/clocks generated by a lower-level drive planification framework, and the ability to qualify and combine findings of constraints to feed others is genuinely original in our approach (we believe). On this latter point, a comparison with the Functional Safety domain of (timed) Fault Trees could be in order here.

VII. Conclusion And Future Work

In this paper, we attempted to illustrate the use of Multiform Logical Time in the context of automated driving for modeling safety. We put forward a framework to monitor the safety of the trajectory plan. In this framework, we present a safe specification pattern to build the safe driving rules specification for different scenarios. In the specification, space constraints are based on primitive events/clocks that are generated by encounters of well-defined area trajectories, thereby combining time and space constraints by CCSL. We also present an efficient approach to irregularly update the constraints in the specification.

Much remains to be done to continue our efforts, such as generating the safe driving rules specification automatically for different scenarios with our safe specification pattern, monitoring the trajectory plan with the spatial events generated by interaction on area trajectories.

References

[1] A. Ziebinski, R. Cupek, D. Grzechca, and L. Chruszczyk, "Review of advanced driver assistance systems (adas)," in *AIP Conference Proceedings*, vol. 1906, no. 1. AIP Publishing LLC, 2017, p. 120002.

[2] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.

[3] T. Economist, "Why uber's self-driving car killed a pedestrian," 2017.

[4] V. Ciancia, G. Grilletti, D. Latella, M. Loreti, and M. Massink, "An experimental spatio-temporal model checker," in *SEFM 2015 Collocated Workshops*. Springer, 2015, pp. 297–311.

[5] D. Gabelaia, R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev, "Combining spatial and temporal logics: expressiveness vs. complexity," *Journal of Artificial Intelligence Research*, vol. 23, pp. 167–243, 2005.

[6] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink, "Qualitative and quantitative monitoring of spatio-temporal properties," in *Runtime Verification*. Springer, 2015, pp. 21–37.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[8] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[10] B. Ltd., "Apollo," https://apollo.auto/.

[11] C. Ltd., "Webots for automobiles," https://cyberbotics.com/doc/automobile/sumo-interface.

[12] C. André, "Syntax and semantics of the clock constraint specification language (ccsl)," Ph.D. dissertation, INRIA, 2008.

[13] F. Mallet and J.-V. Millo, "Boundness issues in ccsl specifications," in *International Conference on Formal Engineering Methods*. Springer, 2013, pp. 20–35.

[14] M. Zhang, F. Dai, and F. Mallet, "Periodic scheduling for MARTE/CCSL: Theory and practice," *Science of Computer Programming*, vol. 154, pp. 42–60, 2018.

[15] C. André, F. Mallet, and M.-A. Peraldi-Frati, "Multiform time in uml for real-time embedded applications," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*. IEEE, 2007, pp. 232–240.

[16] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[17] C. André and F. Mallet, "Specification and verification of time requirements with ccsl and esterel," in *Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, 2009, pp. 167–176.

[18] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *International Conference on Protocol Specification, Testing and Verification*. Springer, 1995, pp. 3–18.

[19] "Towards standardization of av safety: C++ library for responsibility sensitive safety."

[20] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2019, pp. 1–11.

[21] A. Bakhirkin, T. Ferrère, and O. Maler, "Efficient parametric identification for stl," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, 2018, pp. 177–186.

[22] M. Mehrabian, M. Khayatian, A. Shrivastava, J. C. Eidson, P. Derler, H. A. Andrade, Y.-S. Li-Baboud, E. Griffor, M. Weiss, and K. Stanton, "Timestamp temporal logic (ttl) for testing the timing of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–20, 2017.

[23] T. Bourke and M. Pouzet, "Zélus: A synchronous language with odes," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, 2013, pp. 113–118.

[24] F. Mallet and R. De Simone, "Correctness issues on MARTE/CCSL constraints," *Science of Computer Programming*, vol. 106, pp. 78–92, 2015.

[25] OMG, "UML Profile for MARTE: Modeling and analysis of real-time embedded systems," (November 2009) formal/2009-11-02.

[26] C. André, F. Mallet, and R. De Simone, "Modeling time (s)," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 559–573.

[27] L. Yin, J. Liu, Z. Ding, F. Mallet, and R. De Simone, "Schedulability analysis with CCSL specifications," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1. IEEE, 2013, pp. 414–421.

[28] F. Mallet and M. Zhang, "From logical time scheduling to real-time scheduling," in *39th IEEE Real-Time Systems Symposium*, 2018.

[29] Y. Zhang, Y. Chen, and F. Mallet, "A verification framework for spatio-temporal consistency language with CCSL as a specification language," *Frontiers of Computer Science*, pp. 1–25, 2018.

[30] L. Huang, T. Liang, and E.-Y. Kang, "Formal verification of dynamic and stochastic behaviors for automotive systems," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2019, pp. 11–20.

[31] B. Jeannet and F. Gaucher, "Debugging embedded systems requirements with stimulus: an automotive case-study," 2016.